Automated Reverse-Engineering of a Cloud API

Stéphanie Challita, PhD

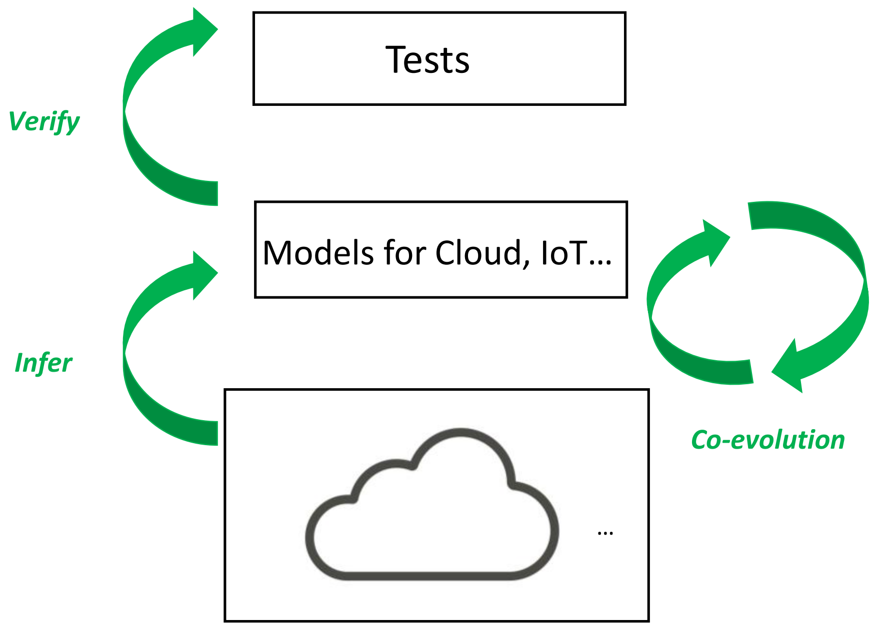
Associate Professor @University of Rennes 1, ESIR & IRISA/Inria, DiverSE team

Brief bio

- **Degree of Systems and Networks Engineering, 2010 - 2015**
- **Research Master's degree in Computer Science, 2014 - 2015**
- **PhD in Computer Science, 2015 - 2018**
  - Spirals team (Lille, CRIStAL, Inria)
- **Postdoctoral Researcher, 2019 - 2020**
  - Kairos team (Sophia Antipolis, I3S, Inria)
- **Associate Professor, since September 2020**
  - DiverSE team (Rennes, IRISA, Inria)

Research project

Towards the automatic construction of reliable and co-evolving software systems



Scientific challenges

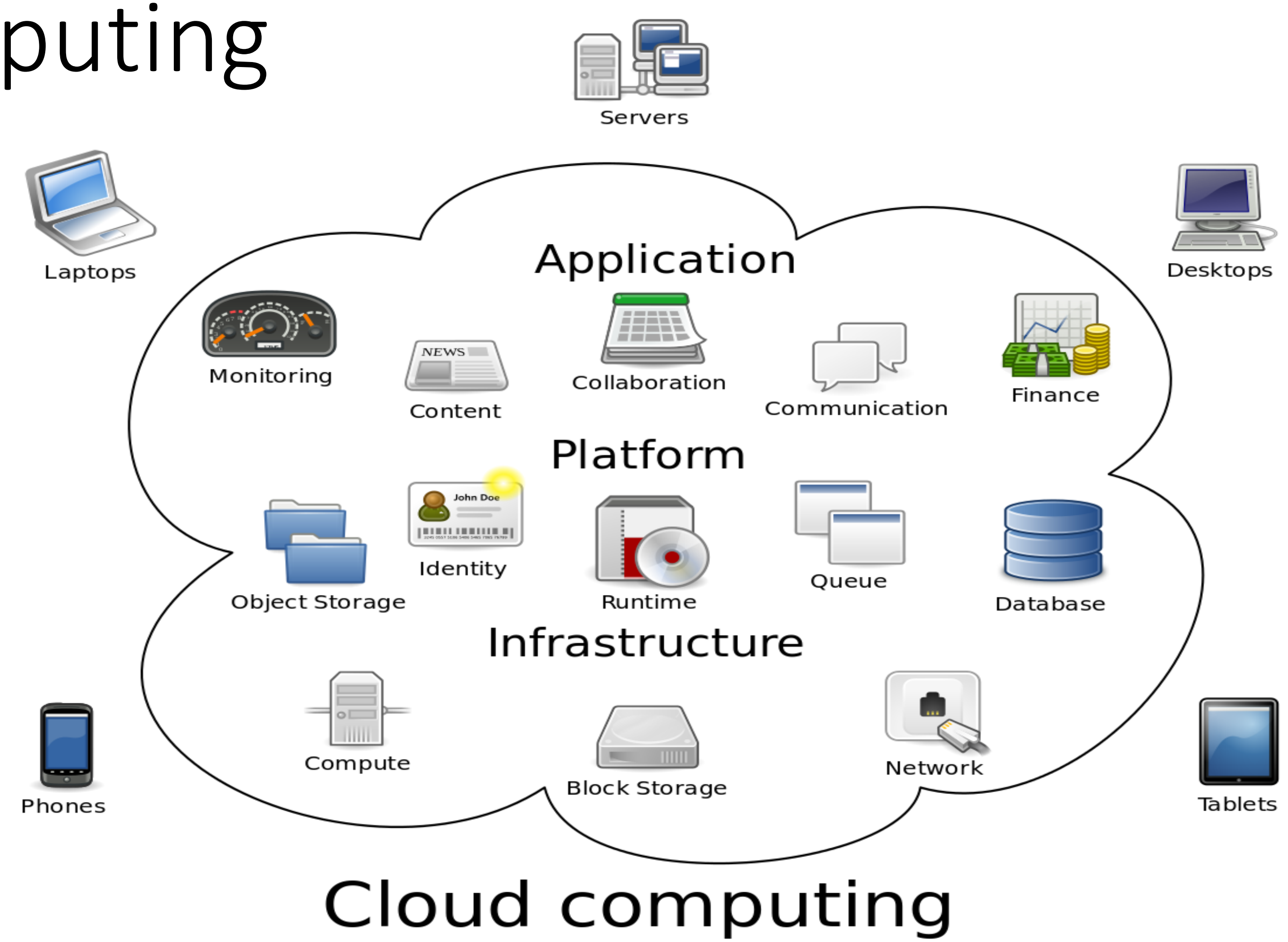
- **Inference of Domain-Specific Modeling Languages (DSML) from APIs**
  - Precision & Genericity
  - Learning
- **Verification**
  - Generation of instances of inferred DSMLs
  - Generation of oracles
- **Co-evolution** of APIs and languages
  - Identify the impact of API changes on DSML
  - Co-evolve the impacted components

Research team

https://www.diverse-team.fr/

- Modeling & Language Engineering
- Advanced testing
- DevOps for distributed and heterogeneous systems
- Variability Engineering

Cloud computing



Created by Sam Johnston, downloaded from https://en.wikipedia.org/wiki/Cloud_computing

Multi-cloud computing

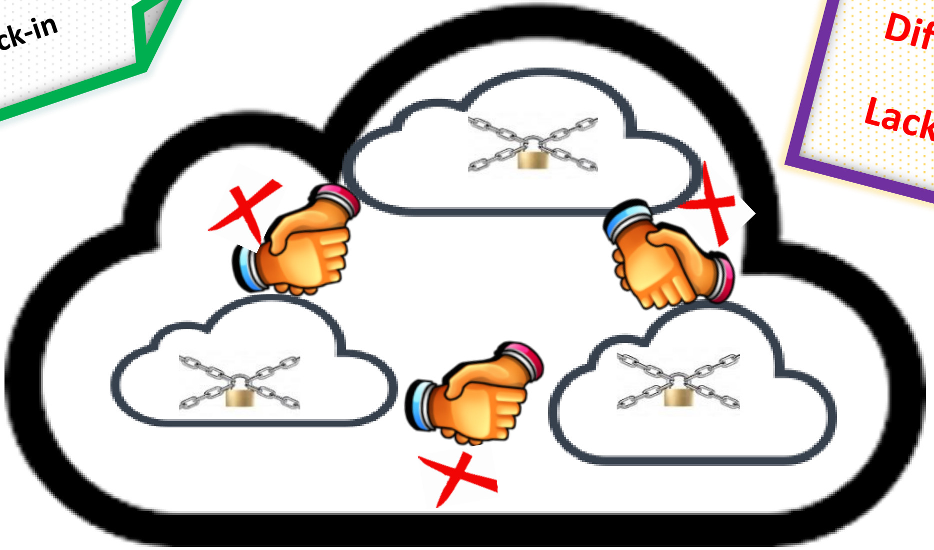
Multi-clouds ensures:

- ✓ Increase performance
- ✓ Reduce costs
- ✓ Ensure resiliency during outages

Multi-cloud computing

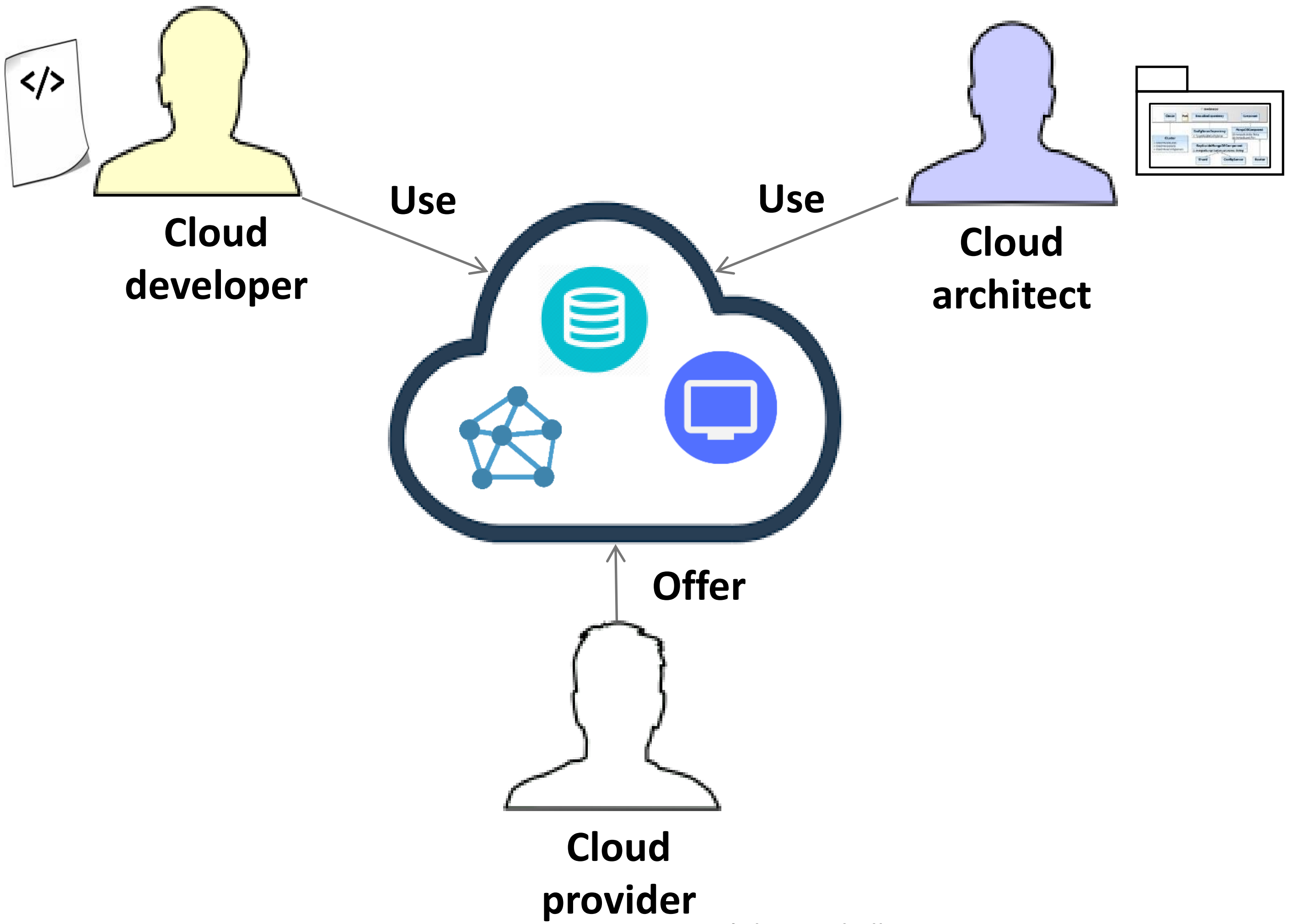
Multi-clouds ensures:

- ✓ Increase performance
- ✓ Reduce costs
- ✓ Ensure resiliency during outages
- ✗ No interoperability
- ✗ Vendor lock-in

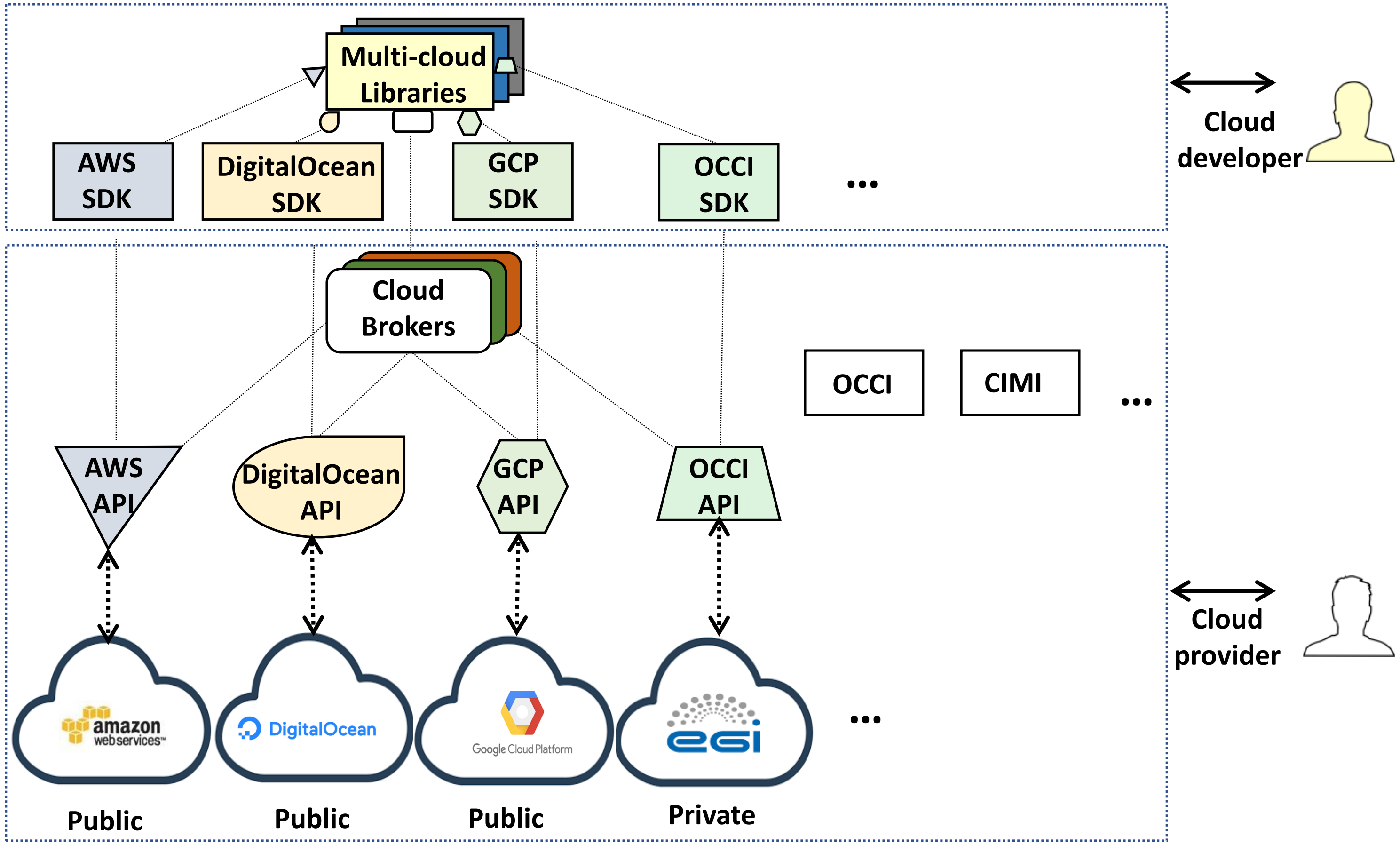


- Different deployment models
- Different management interfaces
- Different service models
- Different semantics
- Lack of formalization

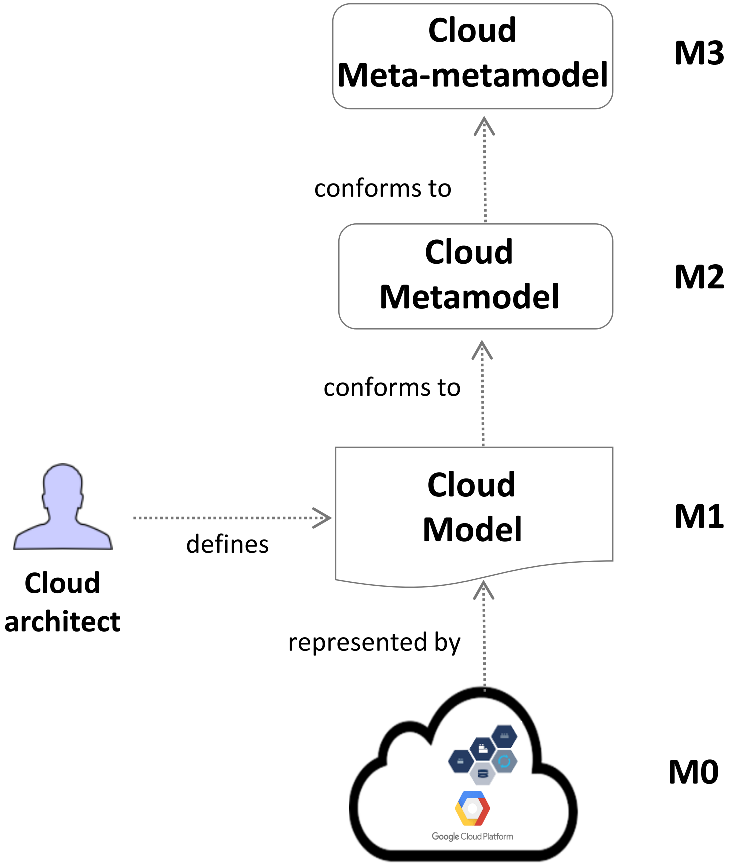
Approaches for multi-clouds - Actors

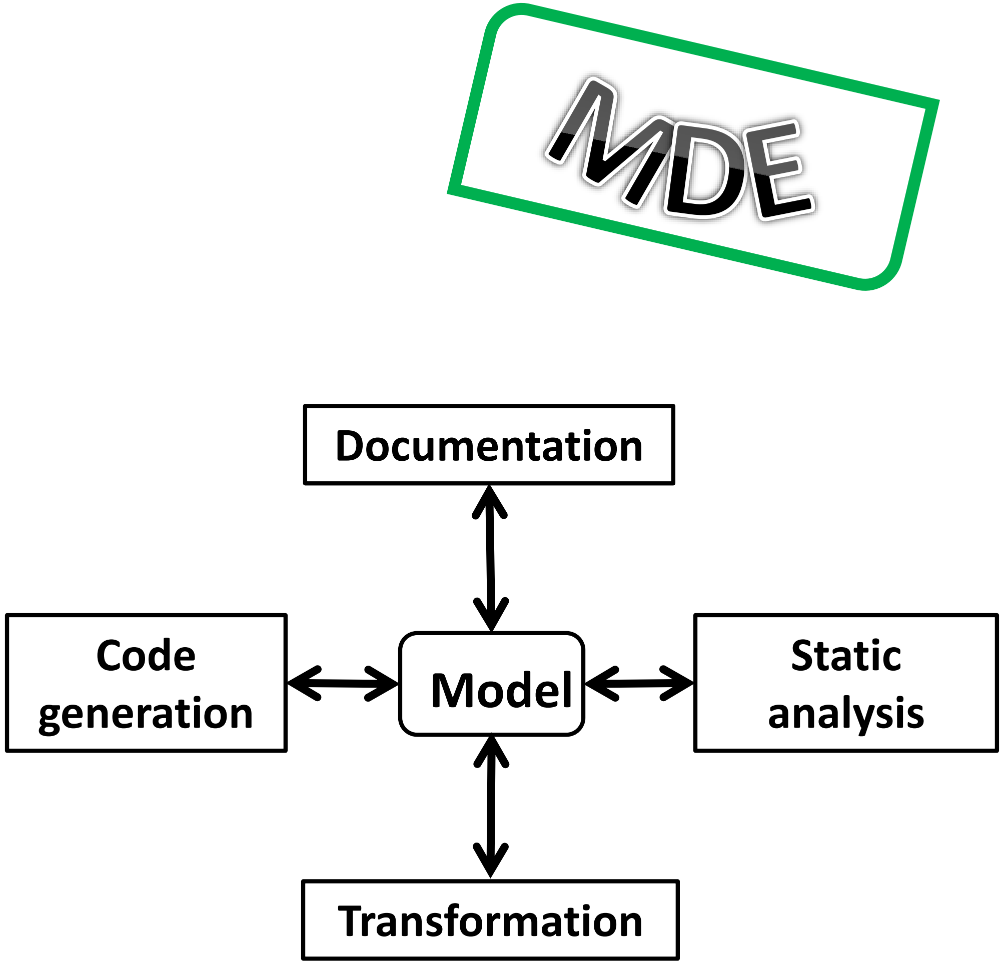


Approaches for multi-clouds

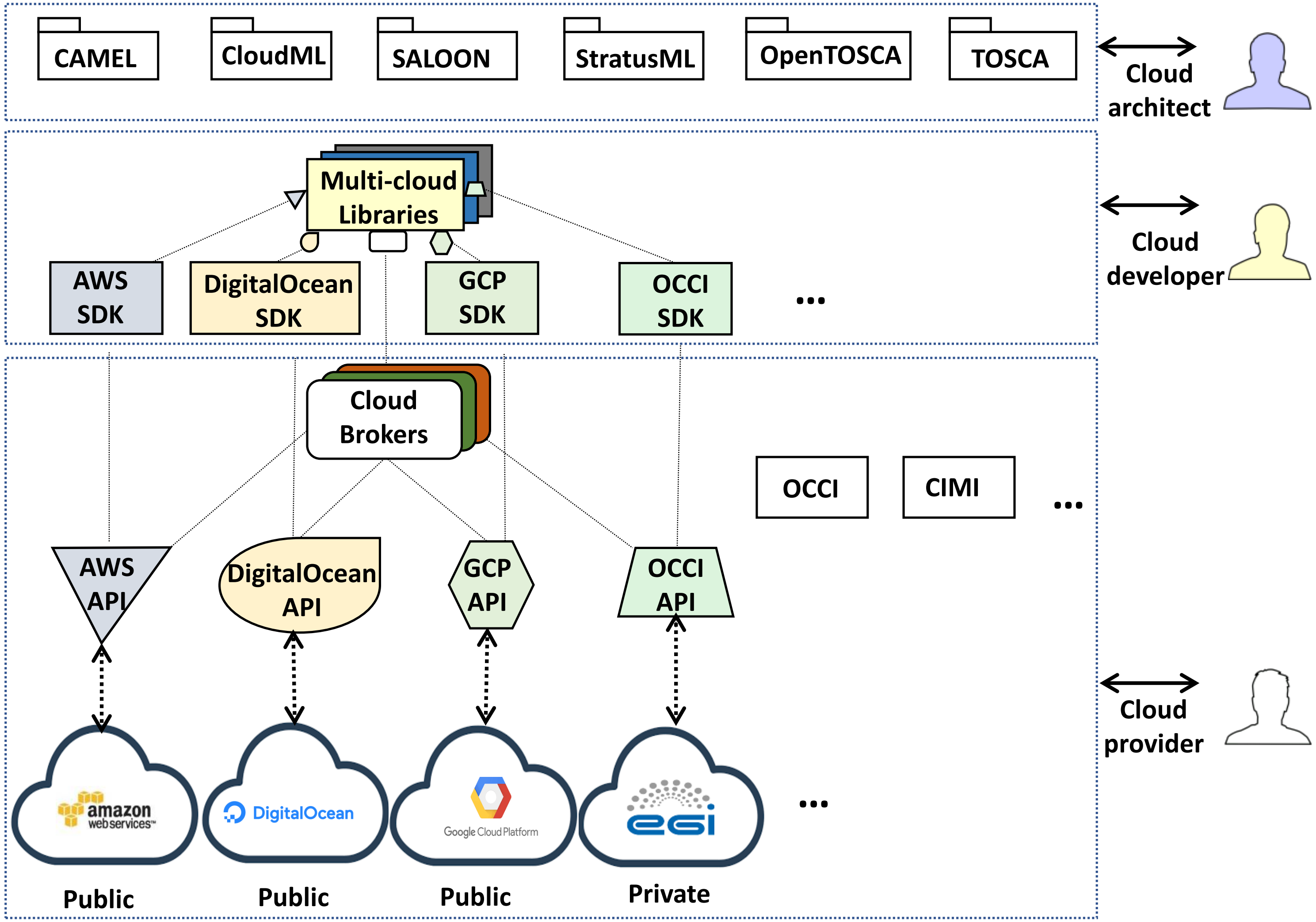


Approaches for multi-clouds





Approaches for multi-clouds



Approaches for multi-clouds

Issue:

Fuzziness of the concepts of the cloud modeling languages

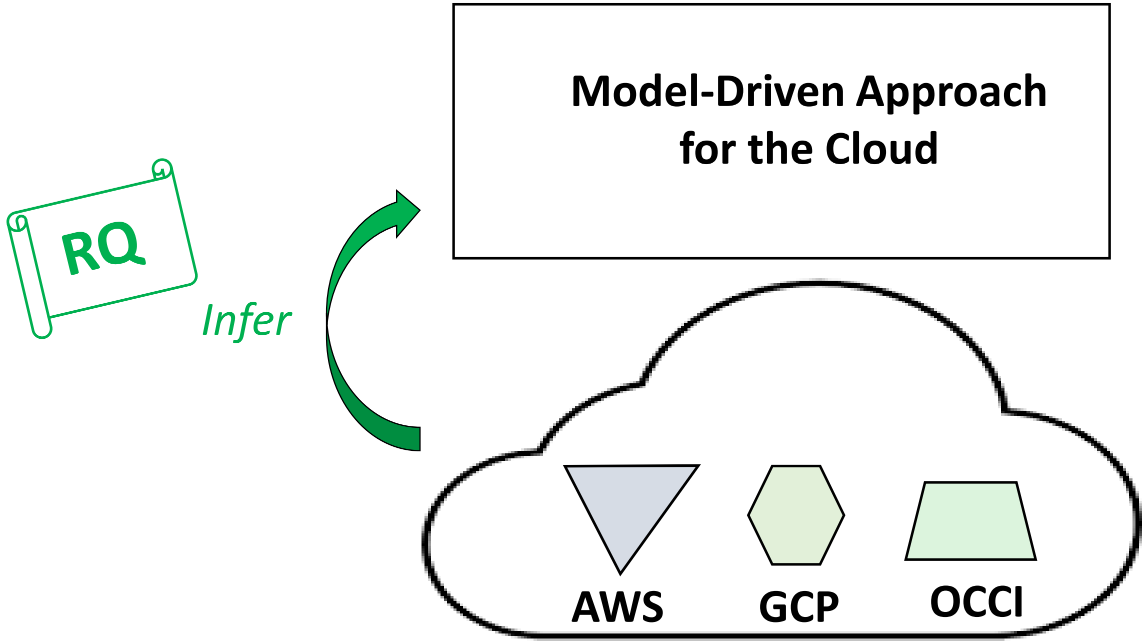
Research question

RQ: *Is it possible to automatically extract precise models from cloud APIs and to synchronize them with the cloud evolution?*

- How to **provide an accurate description** for a cloud API?
- How to **correct** the existing drawbacks in a cloud API documentation?
- How to **analyze** a cloud API documentation?

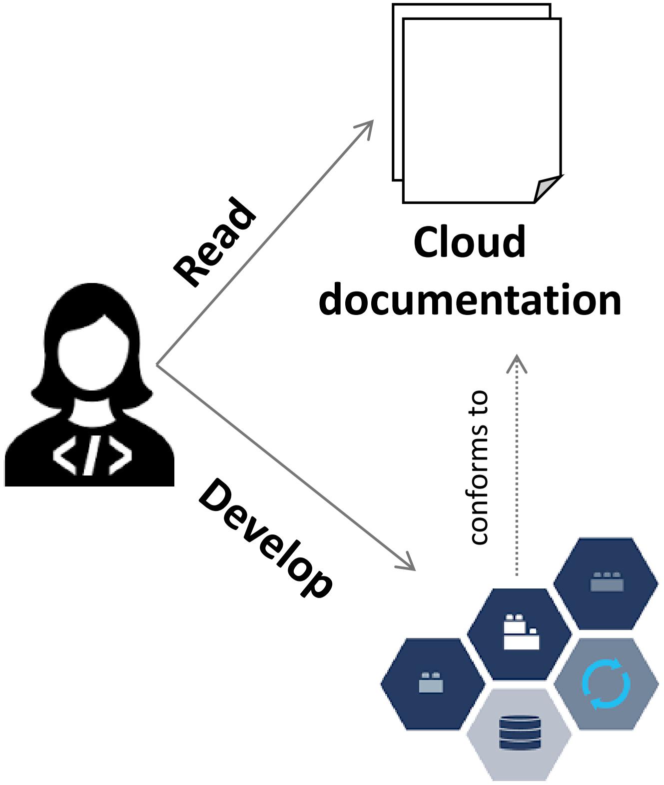
Research topics: API mining, reverse-engineering, NLP

Global vision



Cloud API Documentation

An agreement with the developer on exactly how the system will operate

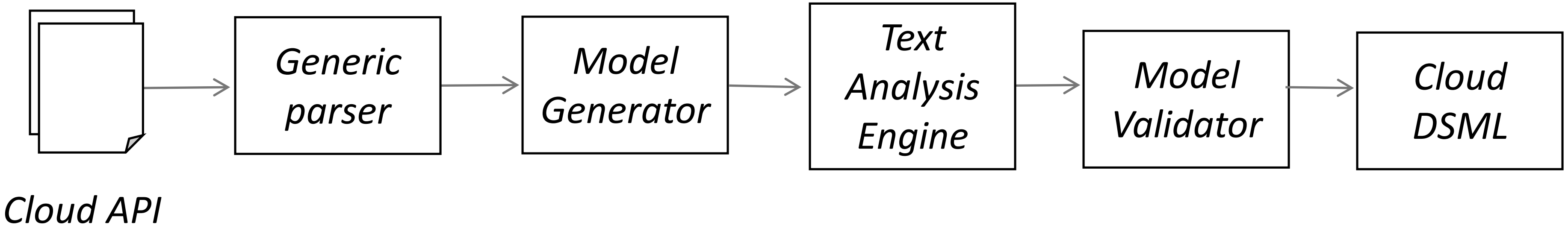


Cloud developer/architect **Cloud provider**

Cloud documentations are written in natural language
→ human errors and/or semantic confusions

Vision

- Inferring models from cloud APIs
  - Work of API mining, reverse-engineering
    - HTML Model
  - Model refinement (NLP techniques, graphical output…)



Google Cloud Platform (GCP) use case

Google Cloud Platform

Is adopted by

Coca-Cola
AIRBUS
htc
Spotify
BNP PARIBAS
MOTOROLA

Is partner with

intel
redhat LINUX
EQUINIX

List of GCP documentation drawbacks

- Informal heterogeneous documentation
- Imprecise types
- Implicit attribute metadata
- Hidden links
- Redundancy
- Lack of visual support

Imprecise types

selfLink	string	[Output Only] Server-defined URL for the resource.

Available at
https://cloud.google.com/compute/docs/reference/latest/targetHttpsProxies

email	string	The email address of the service account. Note: This field is used in responses only. Any value specified here in a request is ignored.

Available at
https://cloud.google.com/iam/reference/rest/v1/projects.serviceAccounts

instanceClass	string	Instance class that is used to run this version. Valid values are: • AutomaticScaling: F1, F2, F4, F4_1G • ManualScaling or BasicScaling: B1, B2, B4, B8, B4_1G Defaults to F1 for AutomaticScaling and B1 for ManualScaling or BasicScaling.

Available at
https://cloud.google.com/appengine/docs/admin-api/reference/rest/v1beta5/apps.services.versions

locations[]	string	The list of Google Compute Engine locations in which the cluster's nodes should be located.

Available at
https://cloud.google.com/container-engine/reference/rest/v1/projects.zones.clusters

Implicit attribute metadata

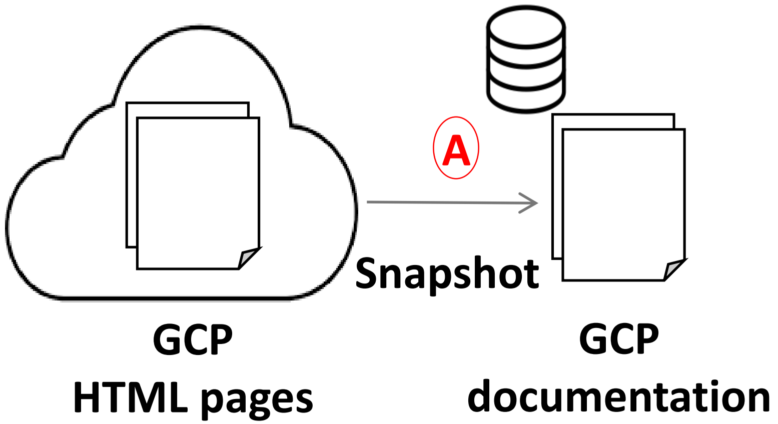
id	unsigned long	[Output Only] The unique identifier for the resource. This identifier is defined by the server.

Available at
https://cloud.google.com/compute/docs/reference/latest/networks

location	string	The geographic location where the dataset should reside. Possible values include EU and US. The default value is US.

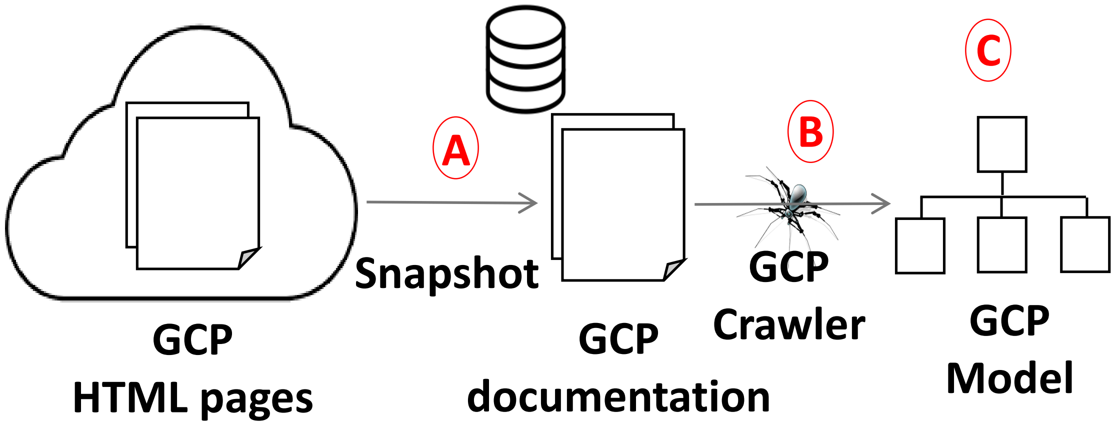
Available at
https://cloud.google.com/bigquery/docs/reference/rest/v2/datasets

GCP snapshot



- GCP engineers could update/correct GCP documentation
- Continuously following up with GCP documentation is costly
- Snapshot of GCP API

GCP crawler & GCP model



- **GCP Crawler** to extract all GCP resources, their attributes and actions
- **GCP Model** for a better description of the GCP resources

GCP crawler & GCP model

Algorithm: Crawler and model generator

Input: Documentation's URL: *url*

Output: Model conforms to OCCI: *model*

```
model ← OCCIFactory.createModel()
document ← connect(url)
for resource in document do
    resourceHTML ← connect(resource.url)
    resourceOCCI ← OCCIFactory.createResource()
    resourceOCCI.properties ← extract(resourceHTML)
    for attributeHTML in resourceHTML do
        attributeOCCI ← OCCIFactory.createAttribute()
        attributeOCCI.properties ← extract(attributeHTML)
        resourceOCCI.attributes.add(attributeOCCI)
    model.append(resourceOCCI)
return model
```

```html
<table>
    <tr>
        <td>bigquery</td>
        <td>
            <a href="docs/reference/bigquery.html">
                Documentation
            </a>
        </td>
    </tr>
    <tr>
        <td>compute</td>
        <td>
            <a href="docs/reference/compute.html">
                Documentation
            </a>
        </td>
    </tr>
    <tr>
        <td>router</td>
        <td>
            <a href="docs/reference/router.html">
                Documentation
            </a>
        </td>
    </tr>
</table>
```

GCP crawler & GCP model

Algorithm: Crawler and model generator

Input: Documentation's URL: *url*

Output: Model conforms to OCCI: *model*

$model \leftarrow OCCIFactory.createModel\,()$
$document \leftarrow connect\,(url)$
for $resource$ in $document$ **do**
    $resourceHTML \leftarrow connect\,(resource.url)$
    $resourceOCCI \leftarrow OCCIFactory.createResource\,()$
    $resourceOCCI.properties \leftarrow extract\,(resourceHTML)$
    for $attributeHTML$ in $resourceHTML$ **do**
        $attributeOCCI \leftarrow OCCIFactory.createAttribute\,()$
        $attributeOCCI.properties \leftarrow extract\,(attributeHTML)$
        $resourceOCCI.attributes.add\,(attributeOCCI)$
    $model.append\,(resourceOCCI)$
return $model$

```html
<table>
    <tr>
        <td>bigquery</td>
        <td>
            <a href="docs/reference/bigquery.html">
                Documentation
            </a>
        </td>
    </tr>
    <tr>
        <td>compute</td>
        <td>
            <a href="docs/reference/compute.html">
                Documentation
            </a>
        </td>
    </tr>
    <tr>
        <td>router</td>
        <td>
            <a href="docs/reference/router.html">
                Documentation
            </a>
        </td>
    </tr>
</table>
```

GCP crawler & GCP model

Algorithm: Crawler and model generator

Input: Documentation's URL: *url*

Output: Model conforms to OCCI: *model*

```
model ← OCCIFactory.createModel ()
document ← connect (url)
for resource in document do
    resourceHTML ← connect (resource.url)
    resourceOCCI ← OCCIFactory.createResource ()
    resourceOCCI.properties ← extract (resourceHTML)
    for attributeHTML in resourceHTML do
        attributeOCCI ← OCCIFactory.createAttribute ()
        attributeOCCI.properties ← extract (attributeHTML)
        resourceOCCI.attributes.add (attributeOCCI)
    model.append (resourceOCCI)
return model
```

```html
<table>
    <tr>
        <td>bigquery</td>
        <td>
            <a href="docs/reference/bigquery.html">
                Documentation
            </a>
        </td>
    </tr>
    <tr>
        <td>compute</td>
        <td>
            <a href="docs/reference/compute.html">
                Documentation
            </a>
        </td>
    </tr>
    <tr>
        <td>router</td>
        <td>
            <a href="docs/reference/router.html">
                Documentation
            </a>
        </td>
    </tr>
</table>
```

GCP crawler & GCP model

Algorithm: Crawler and model generator

Input: Documentation's URL: *url*

Output: Model conforms to OCCI: *model*

```
model ← OCCIFactory.createModel()
document ← connect(url)
for resource in document do
    resourceHTML ← connect(resource.url)
    resourceOCCI ← OCCIFactory.createResource()
    resourceOCCI.properties ← extract(resourceHTML)
    for attributeHTML in resourceHTML do
        attributeOCCI ← OCCIFactory.createAttribute()
        attributeOCCI.properties ← extract(attributeHTML)
        resourceOCCI.attributes.add(attributeOCCI)
    model.append(resourceOCCI)
return model
```

```
<table>
    <tr>
        <td>bigquery</td>
        <td>
            <a href="docs/reference/bigquery.html">
                Documentation
            </a>
        </td>
    </tr>
    <tr>
        <td>compute</td>
        <td>
            <a href="docs/reference/compute.html">
                Documentation
            </a>
        </td>
    </tr>
    <tr>
        <td>router</td>
        <td>
            <a href="docs/reference/router.html">
                Documentation
            </a>
        </td>
    </tr>
</table>
```

GCP crawler & GCP model

Algorithm: Crawler and model generator

Input: Documentation's URL: *url*
Output: Model conforms to OCCI: *model*

$model \leftarrow OCCIFactory.createModel\,()$
$document \leftarrow connect\,(url)$
for $resource$ in $document$ **do**
  $resourceHTML \leftarrow connect\,(resource.url)$
  $resourceOCCI \leftarrow OCCIFactory.createResource\,()$
  $resourceOCCI.properties \leftarrow extract\,(resourceHTML)$
  for $attributeHTML$ in $resourceHTML$ **do**
    $attributeOCCI \leftarrow OCCIFactory.createAttribute\,()$
    $attributeOCCI.properties \leftarrow extract\,(attributeHTML)$
    $resourceOCCI.attributes.add\,(attributeOCCI)$
  $model.append\,(resourceOCCI)$
return $model$

```html
<h3>BigQuery</h3>
<p>
    Google BigQuery is a data warehouse [...] queries
</p>
<table>
    <tr>
        <!-- name -->
        <td>location</td>
        <!-- type -->
        <td>string</td>
        <!-- descr -->
        <td>
            The geographic [...] The default value is US.
        </td>
    </tr>
    <tr>
        <!-- name -->
        <td>lastModifiedTime</td>
        <!-- type -->
        <td>long</td>
        <!-- descr -->
        <td>
            [Output-only] The date [...] the epoch.
        </td>
    </tr>
</table>
```

GCP crawler & GCP model

Algorithm: Crawler and model generator

Input: Documentation's URL: *url*
Output: Model conforms to OCCI: *model*

$model \leftarrow OCCIFactory.createModel\,()$
$document \leftarrow connect\,(url)$
for $resource$ in $document$ **do**
  $resourceHTML \leftarrow connect\,(resource.url)$
  $resourceOCCI \leftarrow OCCIFactory.createResource\,()$
  $resourceOCCI.properties \leftarrow extract\,(resourceHTML)$
  for $attributeHTML$ in $resourceHTML$ **do**
    $attributeOCCI \leftarrow OCCIFactory.createAttribute\,()$
    $attributeOCCI.properties \leftarrow extract\,(attributeHTML)$
    $resourceOCCI.attributes.add\,(attributeOCCI)$
  $model.append\,(resourceOCCI)$
return $model$

```html
<h3>BigQuery</h3>
<p>
    Google BigQuery is a data warehouse [...] queries
</p>
<table>
    <tr>
        <!-- name -->
        <td>location</td>
        <!-- type -->
        <td>string</td>
        <!-- descr -->
        <td>
            The geographic [...] The default value is US.
        </td>
    </tr>
    <tr>
        <!-- name -->
        <td>lastModifiedTime</td>
        <!-- type -->
        <td>long</td>
        <!-- descr -->
        <td>
            [Output-only] The date [...] the epoch.
        </td>
    </tr>
</table>
```

GCP crawler & GCP model

Algorithm: Crawler and model generator

Input: Documentation's URL: *url*

Output: Model conforms to OCCI: *model*

```
model ← OCCIFactory.createModel()
document ← connect(url)
for resource in document do
    resourceHTML ← connect(resource.url)
    resourceOCCI ← OCCIFactory.createResource()
    resourceOCCI.properties ← extract(resourceHTML)
    for attributeHTML in resourceHTML do
        attributeOCCI ← OCCIFactory.createAttribute()
        attributeOCCI.properties ← extract(attributeHTML)
        resourceOCCI.attributes.add(attributeOCCI)
    model.append(resourceOCCI)
return model
```

```html
<h3>BigQuery</h3>
<p>
    Google BigQuery is a data warehouse [...] queries
</p>
<table>
    <tr>
        <!-- name -->
        <td>location</td>
        <!-- type -->
        <td>string</td>
        <!-- descr -->
        <td>
            The geographic [...] The default value is US.
        </td>
    </tr>
    <tr>
        <!-- name -->
        <td>lastModifiedTime</td>
        <!-- type -->
        <td>long</td>
        <!-- descr -->
        <td>
            [Output-only] The date [...] the epoch.
        </td>
    </tr>
</table>
```

GCP crawler & GCP model

```
Algorithm: Crawler and model generator
Input: Documentation's URL: url
Output: Model conforms to OCCI: model
model ← OCCIFactory.createModel ()
document ← connect (url)
for resource in document do
    resourceHTML ← connect (resource.url)
    resourceOCCI ← OCCIFactory.createResource ()
    resourceOCCI.properties ← extract (resourceHTML)
    for attributeHTML in resourceHTML do
        attributeOCCI ← OCCIFactory.createAttribute ()
        attributeOCCI.properties ← extract (attributeHTML)
        resourceOCCI.attributes.add (attributeOCCI)
    model.append (resourceOCCI)
return model
```

```html
<h3>BigQuery</h3>
<p>
    Google BigQuery is a data warehouse [...] queries
</p>
<table>
    <tr>
        <!-- name -->
        <td>location</td>
        <!-- type -->
        <td>string</td>
        <!-- descr -->
        <td>
            The geographic [...] The default value is US.
        </td>
    </tr>
    <tr>
        <!-- name -->
        <td>lastModifiedTime</td>
        <!-- type -->
        <td>long</td>
        <!-- descr -->
        <td>
            [Output-only] The date [...] the epoch.
        </td>
    </tr>
</table>
```

GCP crawler & GCP model

Algorithm: Crawler and model generator

Input: Documentation's URL: *url*

Output: Model conforms to OCCI: *model*

$model \leftarrow OCCIFactory.createModel\,()$

$document \leftarrow connect\,(url)$

for $resource$ in $document$ **do**

  $resourceHTML \leftarrow connect\,(resource.url)$

  $resourceOCCI \leftarrow OCCIFactory.createResource\,()$

  $resourceOCCI.properties \leftarrow extract\,(resourceHTML)$

  for $attributeHTML$ in $resourceHTML$ **do**

    $attributeOCCI \leftarrow OCCIFactory.createAttribute\,()$

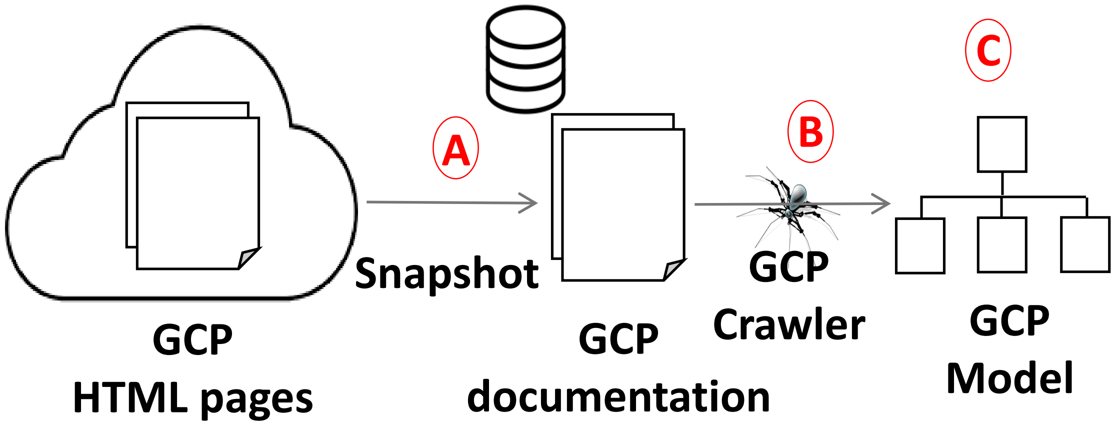
    $attributeOCCI.properties \leftarrow extract\,(attributeHTML)$

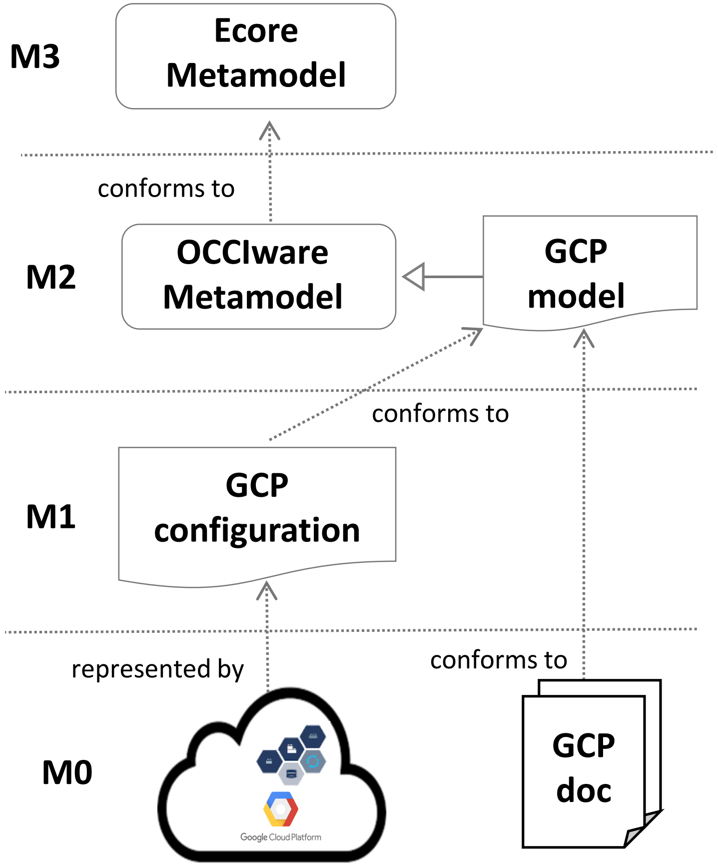
    $resourceOCCI.attributes.add\,(attributeOCCI)$

  $model.append\,(resourceOCCI)$

return $model$

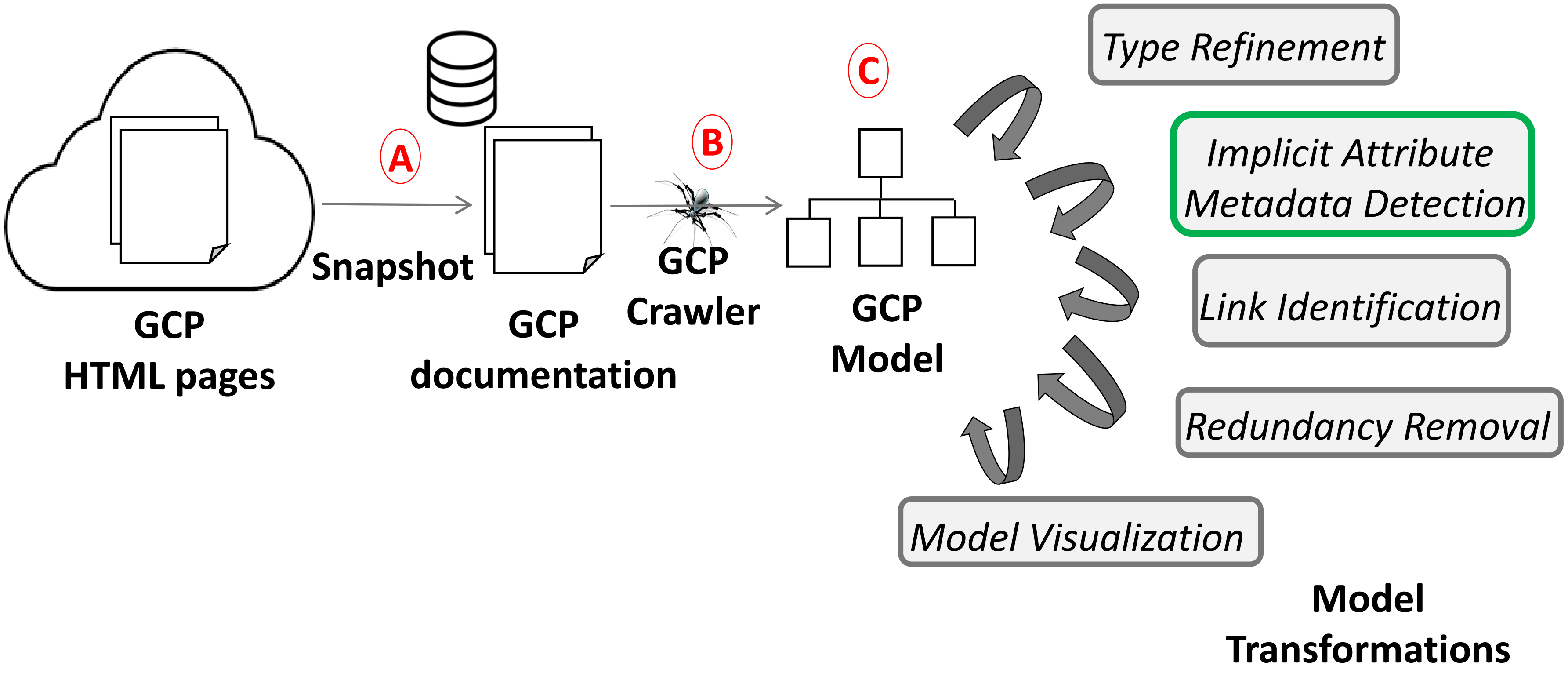
GCP crawler & GCP model





- **GCP Crawler** to extract all GCP resources, their attributes and actions
- **GCP Model** for a better description of the GCP resources

GCP crawler & GCP model



Implicit attribute metadata detection

- To explicitly store information into additional attributes defined in the ATTRIBUTE concept of our GCP MODEL

- We use Natural Language Processing (NLP) techniques
  - Word Tagging/Part-of- Speech (PoS)

- We declare pre-defined tags for some GCP specific attribute properties:
  - mutable = true if [Input-Only]
  - mutable = false if [Output-only]/read only
  - required = true if [Required]
  - required = false if [Optional]
  - default = X if The default value is X

Implicit attribute metadata detection

Algorithm: Implicit attribute metadata detection.

Input: AttributeOCCI: OCCI attribute extracted from HTML page.
Output: AttributeOCCI: OCCI attribute with explicit metadata.

1: $rules \leftarrow \{$"The default value is", "always", "..." $\}$
2: $desc \leftarrow AttributeOCCI.description$
3: **for** $rule$ in $rules$ **do**
4: **if** $desc.contains\,(rule)$ **then**
5: $value \leftarrow rule.getValue\,(desc)$
6: $rule.setValue\,(AttributeOCCI, value)$
7: **return** $AttributeOCCI$

```html
<h3>BigQuery</h3>
<p>
    Google BigQuery is a data warehouse [...] queries
</p>
<table>
    <tr>
        <!-- name -->
        <td>location</td>
        <!-- type -->
        <td>string</td>
        <!-- descr -->
        <td>
            The geographic [...] The default value is US.
        </td>
    </tr>
    <tr>
        <!-- name -->
        <td>lastModifiedTime</td>
        <!-- type -->
        <td>long</td>
        <!-- descr -->
        <td>
            [Output-only] The date [...] the epoch.
        </td>
    </tr>
</table>
```

Implicit attribute metadata detection

Algorithm: Implicit attribute metadata detection.

Input: AttributeOCCI: OCCI attribute extracted from HTML page.
Output: AttributeOCCI: OCCI attribute with explicit metadata.

```
rules ← {"The default value is", "always", "..." }
desc ← AttributeOCCI.description
for rule in rules do
    if desc.contains (rule) then
        value ← rule.getValue (desc)
        rule.setValue (AttributeOCCI, value)
return AttributeOCCI
```

```html
<h3>BigQuery</h3>
<p>
    Google BigQuery is a data warehouse [...] queries
</p>
<table>
    <tr>
        <!-- name -->
        <td>location</td>
        <!-- type -->
        <td>string</td>
        <!-- descr -->
        <td>
            The geographic [...] The default value is US.
        </td>
    </tr>
    <tr>
        <!-- name -->
        <td>lastModifiedTime</td>
        <!-- type -->
        <td>long</td>
        <!-- descr -->
        <td>
            [Output-only] The date [...] the epoch.
        </td>
    </tr>
</table>
```

Implicit attribute metadata detection

Algorithm: Implicit attribute metadata detection.

Input: AttributeOCCI: OCCI attribute extracted from HTML page.
Output: AttributeOCCI: OCCI attribute with explicit metadata.

1: $rules \leftarrow \{$"The default value is", "always", "..." $\}$
2: $desc \leftarrow AttributeOCCI.description$
3: **for** $rule$ in $rules$ **do**
4: **if** $desc.contains\,(rule)$ **then**
5: $value \leftarrow rule.getValue\,(desc)$
6: $rule.setValue\,(AttributeOCCI, value)$
7: **return** $AttributeOCCI$

```
<h3>BigQuery</h3>
<p>
    Google BigQuery is a data warehouse [...] queries
</p>
<table>
    <tr>
        <!-- name -->
        <td>location</td>
        <!-- type -->
        <td>string</td>
        <!-- descr -->
        <td>
            The geographic [...] The default value is US.
        </td>
    </tr>
    <tr>
        <!-- name -->
        <td>lastModifiedTime</td>
        <!-- type -->
        <td>long</td>
        <!-- descr -->
        <td>
            [Output-only] The date [...] the epoch.
        </td>
    </tr>
</table>
```

Implicit attribute metadata detection

Algorithm: Implicit attribute metadata detection.

Input: AttributeOCCI: OCCI attribute extracted from HTML page.
Output: AttributeOCCI: OCCI attribute with explicit metadata.

$rules \leftarrow \{$"The default value is", "always", "..." $\}$
$desc \leftarrow AttributeOCCI.description$
for $rule$ in $rules$ **do**
  if $desc.contains\,(rule)$ **then**
    $value \leftarrow rule.getValue\,(desc)$
    $rule.setValue\,(AttributeOCCI, value)$
return $AttributeOCCI$

```
<h3>BigQuery</h3>
<p>
    Google BigQuery is a data warehouse [...] queries
</p>
<table>
    <tr>
        <!-- name -->
        <td>location</td>
        <!-- type -->
        <td>string</td>
        <!-- descr -->
        <td>
            The geographic [...] The default value is US.
        </td>
    </tr>
    <tr>
        <!-- name -->
        <td>lastModifiedTime</td>
        <!-- type -->
        <td>long</td>
        <!-- descr -->
        <td>
            [Output-only] The date [...] the epoch.
        </td>
    </tr>
</table>
```

Perspectives

- More generic crawler to support any API
  - Do not conform to OCCI anymore, other than HTML pages
- Implement GCP studio, a model-based framework that relies on this approach to design and deploy GCP applications
- Automated approach that would automatically handle the evolution of GCP
  - Incrementally detect streaming modifications, by calculating and modifying only the differences between the initially processed version and the newly modified one

Research internship for M2 students

Research needs you!

Thank you!

Stéphanie Challita, Faiez Zalila, Christophe Gourdin, Philippe Merle. *"A Precise Model for Google Cloud Platform" .* ***IEEE International Conference on Cloud Engineering (IC2E).*** 2018.

https://github.com/occiware/GCP-Model

stephanie.challita@inria.fr

https://stephaniechallita.github.io/